Algorithm NCL for constrained optimization: Solving the linear systems within interior methods

Michael Saunders, Ding Ma, Alexis Montoison, Dominique Orban

Abstract

1 Constrained optimization

We consider large smooth constrained optimization problems of the form

NC	$\min_{x \in \Re^n}$	$\phi(x)$
	subject to	$c(x) = 0, \quad \ell \leq x \leq u,$

where $\phi(x)$ is a smooth scalar function and $c(x) \in \Re^m$ is a vector of smooth linear or nonlinear functions. We assume that first and second derivatives are available. If the constraints include any linear or nonlinear inequalities, we assume that slack variables have already been included as part of x, and appropriate bounds are included in ℓ and u. Problem NC is general in this sense.

2 LANCELOT

LANCELOT [1, 2, 6] is designed to solve large, smooth constrained optimization problems. For problem NC, LANCELOT solves a sequence of about 10 BCL (Bound-Constrained augmented Lagrangian) subproblems of the form

BC_k	$\min_{x \in \Re^n}$	$\phi(x) - y_k^T c(x) + \frac{1}{2}\rho_k c(x)^T c(x)$
	subject to	$\ell \leq x \leq u,$

where y_k is an estimate of the dual variables for the nonlinear constraints $c(x) = 0$, and $\rho_k > 0$ is a penalty parameter. After BC_k is solved (perhaps approximately) to give a subproblem solution x_k^*, the size of $\|c(x_k^*)\|$ is used to define BC_{k+1}:

- If $\|c(x_k^*)\|$ is sufficiently small, stop with "Optimal solution found".
- If $\|c(x_k^*)\| < \|c(x_{k-1}^*)\|$ sufficiently, update $y_{k+1} = y_k - \rho_k c(x_k^*)$ and keep $\rho_{k+1} = \rho_k$.
- Otherwise, keep $y_{k+1} = y_k$ and increase the penalty (say $\rho_{k+1} = 10\rho_k$).
- If the penalty is too large (say $\rho_{k+1} > 10^{10}$), stop with "The problem is infeasible".

3 Algorithm NCL

Algorithm NCL [7] mimics LANCELOT with only one change: subproblem BC_k is replaced by the equivalent larger subproblem

NC_k	$\min_{x \in \Re^n,\, r \in \Re^m}$	$\phi(x) + y_k^T r + \frac{1}{2}\rho_k r^T r$
	subject to	$c(x) + r = 0, \quad \ell \leq x \leq u.$

Given a subproblem solution (x_k^*, r_k^*), the choice between updating y_k or increasing ρ_k is based on $\|r_k^*\|$. We expect $\|r_k^*\| \to 0$, so that x_k^* is increasingly close to solving NC.

The active-set solvers CONOPT [3], MINOS [8], and SNOPT [13] are nominally applicable to NC_k. Their reduced-gradient algorithms would naturally choose r as basic variables, and the x variables would be either superbasic (free to move) or nonbasic (fixed at one of the bounds). However, this is inefficient on large problems unless most bounds are active at the subproblem solution x_k^*.

In contrast, interior methods welcome the extra variables r in NC_k, as explained in [7]:

- The Jacobian of $c(x) + r$ always has full row rank. NCL can therefore solve problems whose solution does not satisfy LICQ (the linear independence constraint qualification). It is also applicable to MPEC problems (Mathematical programming problems with equilibrium constraints).
- The sparse-matrix methods used for each iteration of an interior method are affected very little by the increased matrix size.

4 The linear system in nonlinear interior methods

For simplicity, we assume that the bounds $\ell \le x \le u$ are simply $x \ge 0$. Let y and z be dual variables associated with the constraints $c(x) = 0$ and $x \ge 0$ respectively, and let $X = \text{diag}(x)$, $Z = \text{diag}(z)$. When a nonlinear primal-dual interior method such as IPOPT [4] or KNITRO [5] is applied to NC_k, each search direction is obtained from a linear system of the form

$$\begin{pmatrix} -(H + X^{-1}Z) & & J^T \\ & -\rho_k I & I \\ J & I & \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta r \\ \Delta y \end{pmatrix} = \begin{pmatrix} r_2 \\ r_3 \\ r_1 \end{pmatrix}. \tag{K3}$$

Although this system large, the additional variables δr do not damage the sparsity of the matrix. IPOPT and KNITRO have performed well on problem NC_k as it stands, solving systems (K3).

5 Reducing the size of (K3)

For all NC_k, $\rho_k \ge 1$ (and ultimately $\rho_k \gg 1$), and it is stable to eliminate Δr from (K3) to obtain

$$\begin{pmatrix} -(H + X^{-1}Z) & J^T \\ J & \frac{1}{\rho_k} I \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} r_2 \\ r_1 + \frac{r_3}{\rho_k} \end{pmatrix}, \qquad \Delta r = \frac{1}{\rho_k}(\Delta y - r_3). \tag{K2}$$

If the original problem is convex, $H + X^{-1}Z$ is symmetric positive definite (SPD) and it is possible to eliminate Δy:

$$(H + X^{-1}Z + \rho_k J^T J)\Delta x = -r_2 + J^T(r_3 + \rho_k r_1), \qquad \Delta y = r_3 + \rho_k(r_1 - J\Delta x). \tag{K1}$$

These reductions would require recoding of IPOPT and KNITRO (which is not likely to happen), but they are practical within the nonlinear interior solver MadNLP [11].

6 MadNLP, MadNCL, and GPUs

Algorithm NCL has been implemented as MadNCL [10], using MadNLP [11] as the solver for subproblems NC_k. MadNLP has the option of solving (K2) or (K1) rather than (K3).

For convex problems, system (K2) is symmetric quasidefinite (SQD) [14] and it is practical to use sparse indefinite LDL^{T} factorization. MadNLP implements this option using the cuDSS library [12, 9] to utilize GPUs. Alternatively (and again for convex problems), (K1) is SPD and MadNLP can use the cuDSS sparse Cholesky LDL^{T} factorization (unless $J^T J$ is dense).

Thus, for certain large optimization problems, MadNCL is a solver that employs GPUs and in general is much faster than IPOPT or KNITRO. Numerical results are presented for solving security constrained optimal power flow (SCOPF) problems on GPUs.

References

[1] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: A Fortran Package for Large-scale Nonlinear Optimization (Release A)*. Lecture Notes in Computational Mathematics 17. Springer Verlag, Berlin, Heidelberg, New York, London, Paris and Tokyo, 1992.

[2] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. MOS-SIAM Ser. Optim. 1. SIAM, Philadelphia, 2000.

[3] CONOPT home page. `https://www.gams.com/products/conopt/`.

[4] IPOPT open source optimization software. `https://projects.coin-or.org/Ipopt`, accessed July 12, 2024.

[5] KNITRO optimization software. `https://www.artelys.com/solvers/knitro/`, accessed July 12, 2024.

[6] LANCELOT optimization software. `https://github.com/ralna/LANCELOT`, accessed July 12, 2024.

[7] D. Ma, K. L. Judd, D. Orban, and M. A. Saunders. Stabilized optimization via an NCL algorithm. In M. Al-Baali, Lucio Grandinetti, and Anton Purnama, editors, *Numerical Analysis and Optimization, NAO-IV, Muscat, Oman, January 2017*, volume 235 of *Springer Proceedings in Mathematics and Statistics*, pages 173–191. Springer International Publishing Switzerland, 2018.

[8] MINOS sparse nonlinear optimization solver. `https://www.gams.com/latest/docs/S_MINOS.html`, accessed July 12, 2024.

[9] A. Montoison, *CUDSS.jl: Julia interface for NVIDIA cuDSS*. `https://github.com/exanauts/CUDSS.jl`.

[10] A. Montoison, F. Pacaud, M. A. Saunders, S. Shin, and D. Orban. MadNCL: GPU implementation of algorithm NCL. Working paper on Overleaf, 2024.

[11] A. Montoison, F. Pacaud, S. Shin, et al. MadNLP: a solver for nonlinear programming. `https://github.com/MadNLP/MadNLP.jl`, 2024.

[12] NVIDIA cuDSS (Preview): A high-performance CUDA Library for Direct Sparse Solvers https://docs.nvidia.com/cuda/cudss/index.html.

[13] SNOPT sparse nonlinear optimization solver. http://ccom.ucsd.edu/~optimizers/solvers/snopt/, accessed July 12, 2024.

[14] R. J. Vanderbei. Symmetric quasi-definite matrices. *SIAM J. Optim.*, 5:100–113, 1995.